

# CI: Push build to Git Release

## Initiate the push with 'git push origin [tag]'

In both .travis and appveyor it is possible to initiate the push of a build to your Git repository. These instructions assume that you appveyor and travis accounts and know your way around their services and basic process. Your travis account is used to confirm 'successful' builds of linux and osx, and to create those packages and releases.

## How to push a .Travis 'built' file to a new Github 'Release'

For travis refer to [Petri's post #637](#)

### A. Configure CMake and the Travis file for security and permissions

1. You can use this Dashboard-Tactics\_pi [.travis](#) file as it is. The MacOSX section packages a fully automated installation (FAI). The two build target conditional variables are courtesy from Sean d'Epagnier's repositories.
2. Also changes in Cmake include files are needed for Mac Packaging, in particular [PluginPackage.cmake](#) note the MacOSX entries. Also see [this post and necessary changes by Nereus32](#) (There may be some other changes to cmake needed..Petri?)

```
' - SET(TACTICS_INSTRUMENTS ON) '
IF (TACTICS_INSTRUMENTS)
PROJECT(dashboard_tactics_pi)
SET(PACKAGE_NAME dashboard_tactics_pi)
SET(VERBOSE_NAME DashboardTactics)
SET(TITLE_NAME DashboardTactics)
ELSE (TACTICS_INSTRUMENTS)
PROJECT(dashboard_pi)
SET(PACKAGE_NAME dashboard_pi)
SET(VERBOSE_NAME Dashboard)
SET(TITLE_NAME Dashboard)
ENDIF (TACTICS_INSTRUMENTS)
```

Plus, take the dashboard\_tactics\_pi.pkgproj.in where I modified the key NAME as instructed.

Repeat the build process from zero on an empty build directory and let me know.

## B. Bottom of travis.yml file

Example Bottom section of the travis.yml file. Note: indentation is important in yml files, child parts MUST be indented below the owner. This is shown below by "api\_key:" being the owner and "secure:" being the child, noticed that "secure:" is indented.

```
deploy:
- provider: releases
  api_key:
    secure: "Your encrypted github token here"
  file_glob: true
  file: "$TRAVIS_BUILD_DIR/build/*.{deb,rpm,dmg,txz,pkg,pkg.tar.xz}"
  skip_cleanup: true
  on:
    repo: canne/dashboard_tactics_pi
    tags: true
    all_branches: true
```

If you want another deploy section, like launchpad, you will need to create a new "- provider:" with the text "launchpad" following it. Both "- provider:" labels need to have the same indentation.

## C. Install Travis CLI to encrypt a Github secure public key

1. You need a travis CLI (command line interface) to be able to encrypt the GitHub token. The encrypted string is put in your personal .travis.yml file. Unlike AppVeyor services, there is no on-line encryption service, that's why you need to start with a travis CLI installation
2. [Travis CLI Tool](#) for Windows Developers use [Rubyinstaller/Gem/Travis](#) which still uses 1Gb of disk space but is the easiest alternative.
3. Travis CLI installation instructions for Windows (Suggest use of "Travis CLI Tool" above). When you are sure that you have travis CLI up and working, you can start to create the encrypted key. Without it, deployment to GitHub does not work.
4. Travis CLI installation instructions for Linux are just below. Note that Windows10 users can:
  - Install a Travis CLI Tool as listed above (easiest).
  - Create a [WSL-Windows Subsystem for Linux \(Ubuntu 18.04LTS\) on Win10 Installation](#).
  - or create a Virtual Box and install Ubuntu Linux.

and then from the linux command prompt execute:

```
sudo apt-get update --fix-missing
sudo apt-get install ruby
sudo apt-get install gem
ruby -v
gem install travis -v 1.8.10 --no-rdoc --no-ri
```

```
travis version  
response: 1.8.10.
```

```
225mb disk space used.
```

#### D. Encrypt a Github API Token with Travis encryption

1. Create a Github API token for travis in GitHub here [Developer tools](#). Select all rights to repository "repo" and repository only (that's enough - if it does not work, it is something else). Don't close the window until you have made a copy of your key.
2. Open command window and give following command (in Linux) with travis CLI:

```
echo [Paste GitHub Token Here] | travis encrypt -r  
[username]/[repository_name] --org
```

#### E. Copy and Paste the encrypted key into the Travis secure field

1. The command above produces a very long arbitrary string in format "1234abcde....f".
2. Copy and paste that into .travis.yml in secure field as one single line. Don't let your editor to break it on different lines.
3. Warning: Beware the interweb instructions, which are supposed to modify the .travis.yml on the fly...they mess up the file completely the file. So stick to copy-paste.
4. Save the Travis file.

#### F. Start building the .travis project

1. You will need to turn the project "on" after signing into your travis account. Click on the icon in the top right with the down chevron and select 'Settings', the page will display your repositories under the 'Repositories' tab. Make sure the ones you want to use have the slider button to the right with a tick in the button. If it is to the left and shows a cross it is not active.
2. Remember to change VERSION.cmake patch number and date, an commit and push that too! This is important if you are going to create a release as this will help identify the build objects belonging to the release.
3. Then when you commit and push the .travis.yml file, the travis build will start. It is good to check that both of its build processes do pass faultlessly before the next step.
4. It takes awhile for each commit (Linux and Mac) to build correctly, be patient when it is running.
5. Make sure it built successfully. The project will not be saved because travis does not provide that type of storage space.

#### G. Deploy 'build' to Github

1. Now the project is ready to create a Release in Github. There are two ways to do this.
2. Option 1: Create the new 'Tag' from within Github Release Tab.

- In GitHub create a repository, like v1.2.3 as 'tag', description can be anything, but it best to include the version. This is where Travis and AppVeyor both deploy the built contents.
- When you hit the create button, Travis and AppVeyor will create the 'build' and deploy (the condition flags in the configuration file are define this action) to the Github 'Release' Tag.
- That's why it's important to check before you create the 'Tag' that all builds do pass faultlessly. (Repositories cannot be deleted since they are, in fact defined by tags.)
- If you observe 401 error at the deployment phase in travis logs search in google.com with the following phrase for possible causes “travis github <https://api.github.com/user>: 401 - bad credentials”

### 3. Option 2: Command Line - Make a 'Tag' locally and push it to your remote.

- From your command line create a new tag locally, and then push that tag to your remote repository.
- All the same cautions apply, make sure it builds properly first.

```
OPTION 2: TAG COMMANDS  Make tag locally and push to remote
git tag v1.13.1          <--Create the tag on your local repository
git push origin v1.13.1  <--Push the tag to your remote repository,
travis creates build and pushes.
git push --delete origin v1.13.1 <--You've made a mistake, it fails
to build, remove remote tag.
git tag --delete v1.13.1  <--You've made a mistake, it fails to
build, remove local tag.
Fix the problem and start over.
```

## H. Troubleshooting

When the Travis build does not work the error messages can be obscure and not really help too much. A typical message may be:

Skipping a deployment with the provider because this branch is not permitted:

This is not very helpful as it just says something went wrong not what.

1. Check your whole .travis.yml file. Look at it carefully and compare it to the example given.
2. Make sure you have the security token enclosed in double quotes, i.e. “security/token”
3. Make sure the build process has actually worked for the file types you are trying to transfer to github
4. If you have extra values in your yml file make sure you really know what they do. If you have issues, take all the extraneous values out and see if it works then.
5. If necessary find a very small project and play with that as it will compile much quicker.
6. Make all changes in a branch of the main repository as you will have to do multiple commits to get it working. Once it is working you can make the changes to your main branch(es) and clean up the 'test' branch

## How to push an Appveyor 'built' file to a new Github 'Release'

The use and configuration of Appveyor is similar to Travis in many respects. We use Appveyor to create Windows packages.

### A. Configure Appveyor.yml file for security and permissions

```
# Artifacts Configuration
artifacts: # push all files in directory
  path: build\*.exe
  name: installer

# Deploy to GitHub Releases
deploy:
# description: 'release created by AppVeyor CI'
  provider: GitHub
  auth_token: # '%GitHub_auth_token%'
    secure: VVAVg9a...[put the appveyor encryption of your github public token
here ]...f10SYg0tS
  artifact: installer
  draft: true
  prerelease: true
  tag: $(APPVEYOR_REPO_TAG_NAME) # use pushed Tag or insert version name
  on:
    configuration: Release # Debug contains non-redist MS DLLs
    APPVEYOR_REPO_TAG: true # deploy on tag push only
# branch: master # release from master branch only
```

1. You can also use this Weatherfax\_pi [appveyor.yml](#) file for reference and there are some [in process notes](#) available.
2. Login to your Appveyor Account and Create a [Github API Public Security Token](#) under *New Personal Token*, entering a “Note” similar to *Appveyor\_auth\_token\_[plugin\_name]* and then select a Permissions “Scope” for “REPO” by checking “Repo”.
3. Copy the resulting code to your clipboard and somewhere else, as this is the last time you will have access to the public key.

### B. Encrypt a Github secure public key using Appveyor Encrypt Service

1. Sign into the [Appveyor Encryption Service](#) and paste the public key into the field.
2. Push “Encrypt”. Copy the result to clipboard.

### C. Copy and Paste the encrypted key into the Appveyor secure field

1. Then paste that into the appveyor.yml file after `auth_token:` and `secure:`, where the words

<encrypted GitHub API token here> occur. The encrypted public token can be enclosed by quotes.

2. Save the appveyor.yml file,

## D. Start building the Appveyor project

1. You will need to select or turn the project "on" after signing into your appveyor account.
2. Then when you commit appveyor.yml file, the appveyor 'build' will start. It is good to check that its build processes do pass faultlessly before the next step. Commit and push to the remote repository.
3. Additionally you must remember to change VERSION.cmake patch number and date, an commit and push that!

```
git add appveyor.yml
git commit -a -m "update appveyor.yml"
git push origin master
```

## E. Deploy 'build' to Github

1. It takes awhile for the commit (Windows) to build correctly, be patient when it is running.
2. Make sure it built successfully by checking that the Artifact Tab for the install package file.
3. Now the project is ready to create a Release in Github. There are two ways to do this.
4. Option 1: Create the new 'Tag' from within Github Release Tab.
  - In GitHub create a repository, like v1.2.3 as 'tag', description can be anything, but it best to include the version. This is where Travis and AppVeyor both deploy the built contents.
  - When you hit the create button, Travis and AppVeyor will create the 'build' and deploy (the condition flags in the configuration file are define this action) to the Github 'Release' Tag.
  - That's why it's important to check before you create the 'Tag' that all builds do pass faultlessly. (Repositories cannot be deleted since they are, in fact defined by tags.)
  - If you observe 401 error at the deployment phase in travis logs search in google.com with the following phrase for possible causes "travis github <https://api.github.com/user>: 401 - bad credentials"
5. Option 2: Command Line - Make a 'Tag' locally and push it to your remote.
  - From your command line create a new tag locally, and then push that tag to your remote repository.
  - All the same cautions apply, make sure it builds properly first.

**OPTION 2: TAG COMMANDS** Make tag locally and then push to remote

Examples:

```
git tag v1.9.1-ov42    <--Create the tag on your local repository
git push origin v1.9.1-ov42  <--Push the tag to your remote repository, travis
creates build and pushes.
git push --delete origin v1.9.1-ov42  <--You've made a mistake, it fails to
```

```
build, remove remote tag.
git tag --delete v1.9.1-ov42 <--You've made a mistake, it fails to build,
remove local tag.
Fix the problem and start over.
```

## F. What happens upon Deployment

1. The git push origin v1.9.1-ov42 Tag command is executed.
2. Upon which a new Release & Tag "v1.9.3-ov42-test" gets created in GitHub "Releases", which now contains the results of the - - Appveyor build under the "Assets" label. You should find a file similar to "weatherfax\_pi-1.9.3-ov42-win32.exe".
3. See [https://github.com/rgleason/weatherfax\\_pi/releases/tag/v1.9.3-ov42-test](https://github.com/rgleason/weatherfax_pi/releases/tag/v1.9.3-ov42-test)
4. The Appveyor 'Console Tab' shows: <https://ci.appveyor.com/project/rgleason/weatherfax-pi>
5. The Appveyor "Artifacts Tab" shows the file pushed to Github 'Release':  
<https://ci.appveyor.com/project/rgleason/weatherfax-pi/build/artifacts>

```
Collecting artifacts...
Found artifact 'build\weatherfax_pi-1.9.3-ov42-win32.exe' matching
'build\*.exe' path
Uploading artifacts...
[1/1] build\weatherfax_pi-1.9.3-ov42-win32.exe (721,873 bytes)...100%
Deploying using GitHub provider
Creating "v1.9.3-ov42-test" release for repository
"rgleason/weatherfax_pi" tag "v1.9.3-ov42-test" commit
"05b6418e674b7d722424146c8efe2745a88b635b"...OK
Uploading "weatherfax_pi-1.9.3-ov42-win32.exe" to release assets...OK
Build success
```

G. How to get to these Appveyor Locations? The simplest way to access Appveyor is through Github commits eg:[https://github.com/rgleason/weatherfax\\_pi/commits/master](https://github.com/rgleason/weatherfax_pi/commits/master)

- If appveyor succeeds in creation of an Artifact - green check.
- If appveyor succeeds in creation of an Artifact and a Release (if a new Tag is pushed to remote.) - green check.

Note that VERSION.cmake entries/changes and TAG must be manually coordinated!

## Additional Notes

For more information about setting the auth\_token security please read notes in Appveyor Forum discussions:

- [Push built Artifact .exe to Github "Release" \(opensource, public repos, dev account\) Post 13 by fcgleason on Apr 23, 2019 @ 09:31 PM](#)
- See [Git branch --delete and Git Squash -Effect on Appveyor](#)

It just updates the patch number and updates the Appveyor file so that we can push a new tag to the remote repository and then appveyor will push the windows artifact into the Release (new tag). (This just requires the setup of a personal auth\_token and copying the Yaml encrypted version of that into the appveyor file.)

It's a nice efficient way to make a release with an identical tag. This is what I've done on my repository:

## More information for Appveyor

- [Post #2396](#)
- [Post #2047](#)
- [Post #2425](#)
- [Use of appveyor 'auth\\_token' to push a new github release #134](#)
- [Getting auth token](#)

:

From:  
<https://opencpn.org/wiki/dokuwiki/> - **OpenCPN Manuals**

Permanent link:  
[https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer\\_manual:ci-push-build-to-git](https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual:ci-push-build-to-git)

Last update: **2020/03/09 00:16**

